Fondamenti di Informatica T-1, 2014/2015 – Modulo 2

Prova d'Esame 4A di Giovedì 11 Giugno 2015 – tempo a disposizione 2h

Avvertenze per la consegna: apporre all'inizio di ogni file sorgente un commento contenente i propri dati (**cognome, nome, numero di matricola**) e il **numero** della prova d'esame. Al termine, **consegnare tutti i file sorgenti** e i file contenuti nello StartKit.
Nota: **il main non è opzionale**; i *test* richiesti vanno implementati.
Consiglio: per verificare l'assenza di *warning*, eseguire di tanto in tanto *"Rebuild All"*.

Una famosa libreria offre ai clienti la possibilità di ordinare libri non presenti a scaffale. Gli impiegati della libreria segnano tali ordini in un file di testo di nome "**`ordini.txt`**", ogni ordine su una riga diversa. In particolare, per ogni ordine, scrivono nel seguente modo: la **data** in cui è stato fatto l'ordine (tre interi, nel formato gg/mm/aaaa); a seguire, separato da uno spazio, il **titolo** del libro (una stringa di al più 2047 caratteri utili, contenente spazi); a seguire, separato da un carattere ';', il nome del **cliente** (una stringa di al più 2047 caratteri utili, contenente spazi). Ogni riga del file (compresa l'ultima) è sempre terminata da un new line '\n'. Non è noto a priori quanti ordini siano presenti nel file.
In un secondo file denominato "**`arrivi.txt`**" sono memorizzati i libri appena arrivati; ogni riga del file è dedicata ad un libro: prima vi compare il numero di **copie** (un intero), e a seguire separato da uno spazio il **titolo** del libro; ogni riga è terminata da un carattere newline '\n'.
Si vedano a titolo di esempio i file di testo forniti nello StartKit.

Esercizio 1 – Strutture dati Data, Ordine, e funzioni di lett./scritt. (mod. element.h e ordini.h/.c)

Si definisca un'opportuna struttura dati **`Data`** nel file "element.h", atta a rappresentare la data di un ordine. Si definisca poi una opportuna struttura dati **`Ordine`**, al fine di rappresentare i dati relativi ad un singolo ordine, cioè la data, il titolo del libro, e il nome del cliente. Si definisca poi una struttura dati **`Arrivo`**, contenente le informazioni relative ad un libro appena arrivato (numero di copie e titolo).

Si definisca la funzione:

```
Ordine * leggiOrdini(char * nomeFile, int * dim);
```

che, ricevuto in ingresso il nome di un file contenente l'elenco degli ordini, legga da tale file le informazioni e le restituisca tramite un vettore di strutture dati di tipo **`Ordine`** allocato dinamicamente della dimensione minima necessaria. Tramite l'intero **`dim`** passato per riferimento la funzione deve restituire la dimensione del vettore allocato. In caso di errore nell'apertura del file, la funzione deve restituire un puntatore a NULL, e **`dim`** pari al valore zero. Si supponga per semplicità che ogni riga del file sia "ben formata".

Si definisca la procedura:

```
void stampaOrdini(Ordine * v, int dim);
```

che, ricevuto in ingresso un vettore di strutture dati di tipo **`Ordine`**, e la sua dimensione **`dim`**, stampi a video il contenuto di tali strutture dati.

Si definisca la funzione:

```
list leggiArrivi(char * nomeFile);
```

che, ricevuto in ingresso il nome di un file contenente l'elenco dei libri arrivati, legga da tale file le informazioni e le restituisca tramite una lista di strutture dati di tipo **`Arrivo`**. In caso di errore nell'apertura del file, la funzione deve restituire un puntatore a NULL. Si supponga per semplicità che ogni riga del file sia "ben formata".

Il candidato abbia cura di realizzare nel main opportuni test al fine di verificare il corretto funzionamento delle funzioni di cui sopra. Il candidato abbia cura anche di deallocare eventuale memoria allocata dinamicamente nei test.

Fondamenti di Informatica T-1, 2014/2015 – Modulo 2

Prova d'Esame 4A di Giovedì 11 Giugno 2015 – tempo a disposizione 2h

Esercizio 2 – Ordinamento ed eliminazione dei ripetuti (moduli element.h/.c e ordini.h/.c)

Il candidato definisca poi una procedura:

```
void ordina(Ordine * v, int dim);
```

che, ricevuti in ingresso un vettore di strutture dati di tipo **`Ordine`** e la dimensione di tale vettore, ordini il vettore secondo il seguente criterio: in ordine crescente lessicografico in base al cliente; a parità di cliente, in ordine crescente lessicografico in base al titolo; a parità di titolo, in ordine di data (gli ordini più vecchi vengono prima). A tal fine, il candidato utilizzi uno qualunque degli algoritmi di ordinamento visti a lezione.

Il candidato definisca una funzione:

```
Ordine * ripetuti(Ordine * v, int dim_v, int * dim);
```

che, ricevuto in ingresso un vettore di strutture dati di tipo **`Ordine`**, non necessariamente ordinato, e la sua dimensione **`dim_v`**, restituisca un nuovo vettore allocato dinamicamente di strutture dati di tipo **`Ordine`** (non necessariamente della dimensione minima). Il nuovo vettore dovrà contenere tutti gli ordini, senza ripetizioni. Un ordine è ripetuto se riguarda lo stesso titolo e lo stesso cliente (indipendentemente dalla data). Il nuovo vettore dovrà contenere sempre l'ordine più recente (in termini temporali). Si consiglia a tal scopo di usare la funzione di ordinamento di cui sopra. Tramite il parametro **`dim`** la funzione dovrà restituire la dimensione logica del nuovo vettore.

Esercizio 3 – Estrazione dei libri ancora mancanti (modulo ordine.h/ordine.c)

Si definisca una procedura:

```
void mancanti(Ordine * v, int dim, list arrivi);
```

che, ricevuto in ingresso un vettore di strutture dati di tipo **`Ordine`** e la sua dimensione, e una lista di strutture dati di tipo **`Arrivo`** rappresentante i libri arrivati, stampi a video l'elenco dei libri ancora mancanti, raggruppati per ogni singolo cliente. Per semplicità, si ignorino le quantità disponibili di ogni libro arrivato.

Esercizio 4 – Stampa dei clienti, e de-allocazione memoria (main.c)

Il candidato realizzi nella funzione **`main(…)`** un programma che, usando le informazioni fornite tramite i file di esempio fornito nello StartKit e le funzioni definite agli esercizi precedenti, stampi a video l'elenco dei libri ordinati da clienti, ed ancora mancanti, senza ripetizioni.

Al termine del programma, il candidato abbia cura di de-allocare tutta la memoria allocata dinamicamente, ivi compresa la memoria allocata per le liste, se non in structure sharing (in tal caso, si segnali la situazione con un breve commento nel codice).

Fondamenti di Informatica T-1, 2014/2015 – Modulo 2

Prova d'Esame 4A di Giovedì 11 Giugno 2015 – tempo a disposizione 2h

"element.h":

```
#ifndef _ELEMENT
#define _ELEMENT

#include <stdio.h>
#include <string.h>

#define DIM 2048

typedef struct {
	int gg;
	int mm;
	int aaaa;
} Data;

typedef struct {
	Data data;
	char titolo[DIM];
	char cliente[DIM];
} Ordine;

typedef struct {
	int copie;
	char titolo[DIM];
} Arrivo;

typedef Arrivo element;

int compare(Ordine o1, Ordine o2);
int equals(Ordine o1, Ordine o2);

#endif
```

"element.c":

```
#include "element.h"

int compare(Ordine o1, Ordine o2) {
	int result;

	result = strcmp(o1.cliente, o2.cliente);
	if (result!=0) return result;
	result = strcmp(o1.titolo, o2.titolo);
	if (result!=0) return result;
	result = o1.data.aaaa - o2.data.aaaa;
	if (result!=0) return result;
	result = o1.data.mm - o2.data.mm;
	if (result!=0) return result;
	return o1.data.gg - o2.data.gg;
}

int equals(Ordine o1, Ordine o2) {
	return !strcmp(o1.titolo, o2.titolo) && !strcmp(o1.cliente, o2.cliente);
}
```

Fondamenti di Informatica T-1, 2014/2015 – Modulo 2

Prova d'Esame 4A di Giovedì 11 Giugno 2015 – tempo a disposizione 2h

"list.h"

```
#ifndef LIST_H
#define LIST_H

#include "element.h"

typedef struct      list_element
{
    element value;
    struct list_element  *next;
} item;

typedef  item* list;

typedef  int boolean;

/* PRIMITIVE  */
list emptylist(void);
boolean empty(list);
list cons(element, list);
element head(list);
list tail(list);

void showlist(list l);
void freelist(list l);
//int member(element el, list l);
//list insord_p(element el, list l);

#endif
```

"list.c":

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "list.h"

/* OPERAZIONI PRIMITIVE */
list  emptylist(void)            /* costruttore lista vuota */
{
    return NULL;
}

boolean  empty(list l)     /* verifica se lista vuota */
{
    return (l==NULL);
}

list  cons(element e, list l)
{
  list t;       /* costruttore che aggiunge in testa alla lista */
  t=(list)malloc(sizeof(item));
  t->value=e;
  t->next=l;
  return(t);
}
```

Fondamenti di Informatica T-1, 2014/2015 – Modulo 2

Prova d'Esame 4A di Giovedì 11 Giugno 2015 – tempo a disposizione 2h

```
element head(list l) /* selettore testa lista */
{
  if (empty(l)) exit(-2);
  else return (l->value);
}

list  tail(list l)         /* selettore coda lista */
{
  if (empty(l)) exit(-1);
  else return (l->next);
}


void showlist(list l) {
       element temp;
       if (!empty(l)) {
              temp = head(l);
              printf("%d %s\n", temp.copie, temp.titolo);
              return showlist(tail(l));
       }
       else {
              printf("\n\n");
              return;
       }
}




void freelist(list l) {
       if (empty(l))
              return;
       else {
              freelist(tail(l));
              free(l);
       }
       return;
}
```

Fondamenti di Informatica T-1, 2014/2015 – Modulo 2

Prova d'Esame 4A di Giovedì 11 Giugno 2015 – tempo a disposizione 2h

"ordini.h":

```
#ifndef _ORDINI
#define _ORDINI

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "element.h"
#include "list.h"

// Es. 1
Ordine * leggiOrdini(char * nomeFile, int * dim);
void stampaOrdini(Ordine * v, int dim);
list leggiArrivi(char * nomeFile);

// Es. 2
void ordina(Ordine * v, int dim);
Ordine * ripetuti(Ordine * v, int dim_v, int * dim);

// Es. 3
void mancanti(Ordine * v, int dim, list arrivi);

#endif
```

"ordini.c":

```
#include "ordini.h"

int readField(char buffer[], int dimBuffer, char sep, FILE *f) {
	int i = 0;
	char ch = fgetc(f);
	while (ch != sep && ch != '\n' && ch != EOF && i<dimBuffer-1) {
		buffer[i] = ch;
		i++;
		ch = fgetc(f);
	}
	buffer[i] = '\0';
	return ch;
}

Ordine * leggiOrdini(char * nomeFile, int * dim) {
	FILE * fp;
	Ordine * result = NULL;
	Ordine temp;
	int continua;

	*dim = 0;
	fp = fopen(nomeFile, "rt");
	if (fp != NULL) {
		continua = 1;
		while (continua) {
			if ( fscanf(fp, "%d/%d/%d", &(temp.data.gg), &(temp.data.mm),
&(temp.data.aaaa)) != 3)
				continua = 0;
```

```
                        if (continua && fgetc(fp) != ' ')
                                continua = 0;
                        if (continua && readField(temp.titolo, DIM, ';', fp)!=';')
                                continua = 0;
                        if (continua && readField(temp.cliente, DIM, '\n', fp)!='\n')
                                continua = 0;
                        if (continua)
                                *dim = *dim + 1;
                }
                rewind(fp);
                result = (Ordine*) malloc(sizeof(Ordine) * *dim);
                continua = 1;
                *dim = 0;
                while (continua) {
                        if ( fscanf(fp, "%d/%d/%d", &(temp.data.gg), &(temp.data.mm),
&(temp.data.aaaa)) != 3)
                                continua = 0;
                        if (continua && fgetc(fp) != ' ')
                                continua = 0;
                        if (continua && readField(temp.titolo, DIM, ';', fp)!=';')
                                continua = 0;
                        if (continua && readField(temp.cliente, DIM, '\n', fp)!='\n')
                                continua = 0;
                        if (continua) {
                                result[*dim] = temp;
                                *dim = *dim + 1;
                        }
                }
                fclose(fp);
        }
        return result;
}

void stampaOrdini(Ordine * v, int dim) {
        int i;
        for (i=0; i<dim; i++) {
                printf("%d/%d/%d %s;%s\n", v[i].data.gg, v[i].data.mm, v[i].data.aaaa,
                        v[i].titolo, v[i].cliente);
        }
        return;
}

list leggiArrivi(char * nomeFile) {
        FILE * fp;
        list result = NULL;
        Arrivo temp;

        fp = fopen(nomeFile, "rt");
        if (fp!= NULL) {
                result = emptylist();
                while (fscanf(fp, "%d", &(temp.copie))==1) {
                        fgetc(fp);
                        readField(temp.titolo, DIM, '\n', fp);
                        result = cons(temp, result);
                }
                fclose(fp);
        }
```

```
	return result;
}


void scambia(Ordine *a, Ordine *b) {
	Ordine tmp = *a;
	*a = *b;
	*b = tmp;
}

void bubbleSort(Ordine v[], int n) {
	int i, ordinato = 0;
	while (n>1 && !ordinato) {
		ordinato = 1;
		for (i=0; i<n-1; i++)
			if (compare(v[i], v[i+1])>0) {
				scambia(&v[i],&v[i+1]);
				ordinato = 0;
			}
			n--;
	}
}

void ordina(Ordine * v, int dim) {
	bubbleSort(v, dim);
}

Ordine * ripetuti(Ordine * v, int dim_v, int * dim) {
	Ordine * result;
	int i;

	*dim = 0;
	result = (Ordine *) malloc(sizeof(Ordine) * dim_v);
	ordina(v, dim_v);
	for (i=0; i<dim_v-1; i++)
		if (!equals(v[i], v[i+1])) {
			result[*dim] = v[i];
			*dim = *dim+1;
		}
	result[*dim] = v[i];
	*dim = *dim+1;
	return result;
}


int member(Ordine el, list l) {
	if (empty(l))
		return 0;
	else
		if (strcmp(el.titolo, head(l).titolo)==0)
			return 1;
		else
			return member(el, tail(l));
}

void mancanti(Ordine * v, int dim, list arrivi) {
	int i;
```

```
	ordina(v, dim);
	for (i=0; i<dim; i++) {
		if (!member(v[i], arrivi)) {
			printf("Cliente %s:", v[i].cliente);
			printf("\t%s\n", v[i].titolo);
		}
	}
}
```

"main.c":

```
#include "element.h"
#include "ordini.h"

int main() {
	/* Es. 1 */
	{
		Ordine * ord;
		int dim;
		list arrivi;
		ord = leggiOrdini("ordini.txt", &dim);
		stampaOrdini(ord, dim);
		arrivi = leggiArrivi("arrivi.txt");
		showlist(arrivi);
		freelist(arrivi);
		free(ord);
	}
	/* Es. 2 */
	{
		Ordine * ord;
		Ordine * ord2;
		int dim;
		int dim2;
		ord = leggiOrdini("ordini.txt", &dim);
		ord2 = ripetuti(ord, dim, &dim2);
		stampaOrdini(ord2, dim2);
		free(ord);
		free(ord2);
	}
	/* Es. 3 & 4*/
	{
		Ordine * ord;
		Ordine * ord2;
		int dim;
		int dim2;
		list arrivi;
		ord = leggiOrdini("ordini.txt", &dim);
		ord2 = ripetuti(ord, dim, &dim2);
		arrivi = leggiArrivi("arrivi.txt");
		mancanti(ord2, dim2, arrivi);
		freelist(arrivi);
		free(ord);
		free(ord2);
	}

	return 0;
}
```

Fondamenti di Informatica T-1, 2014/2015 – Modulo 2

Prova d'Esame 4A di Giovedì 11 Giugno 2015 – tempo a disposizione 2h

"ordini.txt":

```
06/06/2015 I Maigret 9;Federico Chesani
06/06/2015 I Maigret 10;Federico Chesani
02/06/2015 L'ora di lezione;Paola Mello
24/05/2015 Corso di logica;Paola Mello
22/04/2015 Introduzione al Prolog;Paola Mello
13/05/2015 Le storie della Pimpa;Carlo Giannelli
14/05/2015 Le storie di Giulio Coniglio;Carlo Giannelli
06/06/2015 I Maigret 9;Federico Chesani
```

"arrivi.txt":

```
2 I Maigret 9
5 Le storie della Pimpa
```